



Flash Memory Summit

Designing FTLs for High Capacity Drives

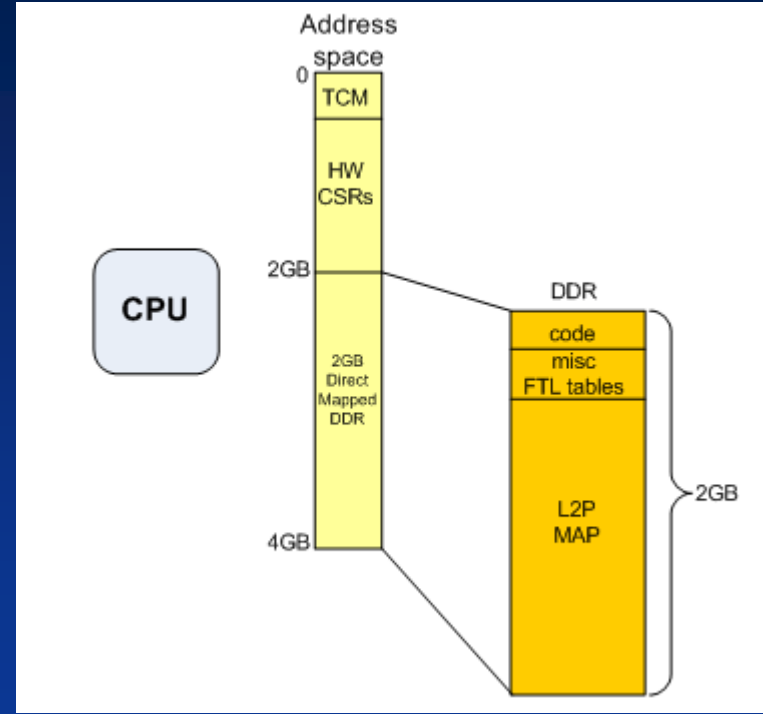
Tim Canepa
Canepa Consulting Group



Today's Drives have it Easy

@ 2TB Capacity

- All the Logical to Physical Address Map fits in CPU reachable memory using 4K map units
- Map Unit addresses are < 32bits so they're easy for the CPU to manipulate
- Recovering 2GB of map even with a flat mapping scheme may be time consuming, but not onerous
- Basically you can make an FTL work with a flat/linear map, a few DMAs, a bunch of CPU horse power and a little IPC





Future Drives will have Challenges

- Drive Capacities are increasing
 - Avg enterprise drives 2-4TB today, go to 16TB+ in 2021
- Most SSDs keep some or all of their FTL mapping structures cached in DDR
- Flash density is currently outpacing DDR density both on individual die and package basis
 - In 2021 timeframe, a 16 die stack using TSV will have 2TB of raw capacity
 - 8 package placements = 16TB while 2 DDR placements may only be 8GB...
- Standards aren't focused on creating more DDR capacity and BW for SSDs
 - WIO, HMC and HBM are all focused on addressing different problems...



Flash Memory Summit

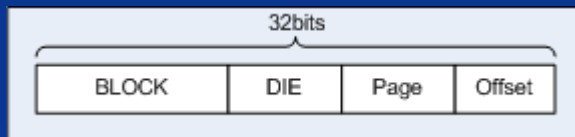
Larger capacity drives have three primary challenges to contend with

- Flash addressing related challenges
- DDR related challenges
- Scaling challenges to FTL design & SSD architecture

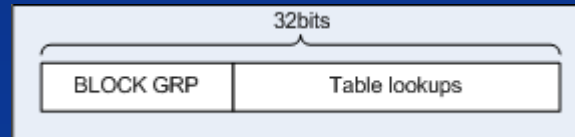


Flash addressing challenges

- Today, most SSDs use a 4K granularity for logical to physical data mapping
 - At current capacities, mapping addresses fit nicely in 32bits
- But with a 4K map unit, 16TB is the max capacity you can address with 32bits
 - And even addressing 16TB using 4K map granularity requires additional “heroics”
 - Variable code rates, non binary page counts, bad blocks/planes/pages require multiple levels of indirection to fit addressing into 32bits
 - So a single map translation could take maybe 5 memory accesses
- So what to do?
 - Larger granularity (8K, 16K, 32K) mapping, or bigger addresses (33bits, 34bits...)
- Why does this matter?
 - Because manipulating unaligned variable size bit fields > 32bits with a CPU is expensive...



Direct addressing

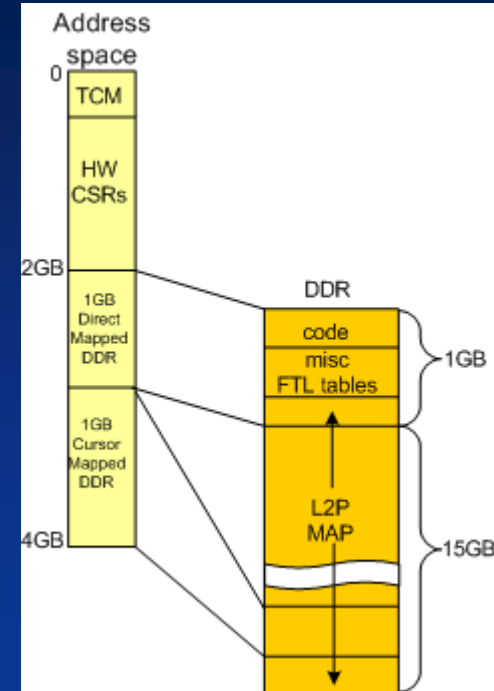


Lookup based addressing



DDR related Challenges

- Achieving large DDR capacities for SSDs creates a host of issues
- Physically connecting enough DDR to hold the entire L2P map is challenging
 - Beyond 2 ranks, transfer rates drop
 - Widening buses to get more connectivity creates routing challenges
 - Just real estate for 8+ placements...
- Addressing >4GB of DDR from a 32bit CPU
 - Need employ cursors or HW assists to access memory
 - And cursors suck as they need to be coherent with memory addressing
 - To be fair, many FTL tables are small and can still fit in CPU address space
 - Or transition to 64bit CPUs
- Any slow down in memory speed will have a non-trivial impact on performance
 - WHY? SSDs typically use a codeword based ECC (64B or larger) to protect DDR
- Bottom Line
 - Even if you can attach enough DDR to hold your entire map @ 4K, you won't likely get the performance you want...



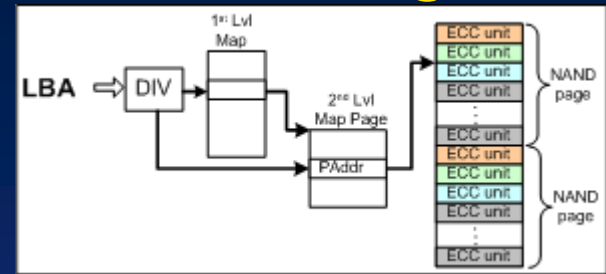
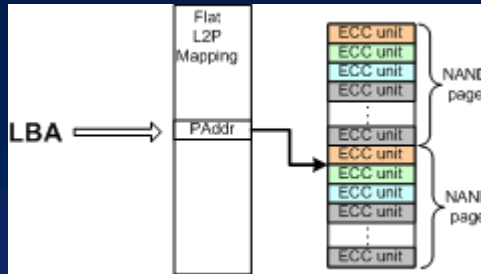


FTL Design Choices Impact...

- Resource Requirements
 - Buffers, memory, hold up energy, HW assists
- Ability to Scale
 - Both up and down
- Write Amplification – including WA bounding
- IOPs & Latency (including tail latencies)
- Time To Ready (TTR)
- Pfail/Suspend/Power Down time
- Low Power capabilities
- Firmware Complexities and Planning



A quick review of basic FTL design Options



- Two basic types of FTL L2P Mapping – Flat & Hierarchical
 - Flat refers to a one level map where the entire map is typically cached in DDR
 - Hierarchical maps have multiple levels, with the upper levels being fully cached in DDR and some or all of the lowest levels being partially cached in DDR
- Pick your L2P Map unit size – typically 4K
- Handling > 4GB of DDR (cursor, assists or 64bit CPU)
- Handling map unit addresses that exceed 32bits (large packed bit fields)
- Handling DDR access overheads (every tbl access consumes 64B of BW)
- Handling a plurality of disparate system and user data velocities
 - Possibly beyond the scope of this discussion, but critical for large capacity drives that are prone to more sharing



Hardware Assists are necessary!

- The combination of large map unit addresses (>32bit) and/or slower DDR along with multiple levels of indirection to decode L2P locations makes HW assists a must...
 - Could be a combination of CPU assists, CPU instruction extensions and CPU translations
- Coherent access and communication in multi-core implementations also need assists
 - CPUs need to communicate, but often memory and peripheral buses are not natively coherent between CPUs, or the coherency mechanisms are expensive so assists are needed
- Each FTL implementation likely needs its own flavor of assists...



Exploring FTL Options to support Large Capacity Drives

- Flat Map with Larger Map Units
- Segmented Flat Map
- Fully Cached Flat Map with 4K map units
- Multiple Chip solution
- Host Based Map
- Multi-Level Map



Flat map with Larger Map Units?









- Simply moving to a larger map unit (8K, 16K, 32K) has a lot of side effects
- Writes & Write Amp
 - Data WA increases linearly as map unit size increases
 - Write overhead and flash bandwidth is impacted by RMW (host writes only)
 - AKA host writes now have a read AND a recycling R/W multiplier
 - Buffering requirements increase somewhat due to tenure induced by RMW
 - Flash bus contention increases
- Reads
 - Added complexity and restrictions to extract data units potentially make reads slower
 - If you don't support reading at map unit offsets, you need to decode the entire map unit to extract 4K of data...
- PFail gets more complicated due to RMW
- Value Prop?
 - Not great – WA reduces warrantee

FTL Attribute	Score
Write Amp	2x plus worse for 4K writes
IOPs & Latency	Reads Fair, Writes Bad
DDR	Good
> 32bit addressing	Good
TTR	Neutral
PFAIL	More Complex due to RMW
FW Complexity	More Complex due to RMW



Segmented Flat Map...

- Break the Flat map into N swappable segments
 - Basically praying for locality
 - Will never work for any kind of random workload
- Random Reads/Writes cause segment swapping
 - Segment swap rate directly related to percentage of map cached
 - Avg Flash data transfer of $(4K * \text{hit rate} + (4K + \text{seg size}) * \text{miss rate})$ for reads alone
 - DDR BW = $\text{seg size} * \text{miss rate}$ for reads alone...
 - Let's just stop here...
- Value Prop?
 - Not great. WA variance and read latency variance can be huge depending on locality

FTL Attribute	Score
Write Amp	 Potentially awful
IOPs & Latency	  Reads fair, Writes Bad
DDR	 Bad in terms of BW
> 32bit addressing	 Needs HW assists
TTR	 Neutral
PFAIL	 More Complex (seg recovery)
FW Complexity	 More complex (seg recovery)



Fully Cached Flat Map with 4K Map Units

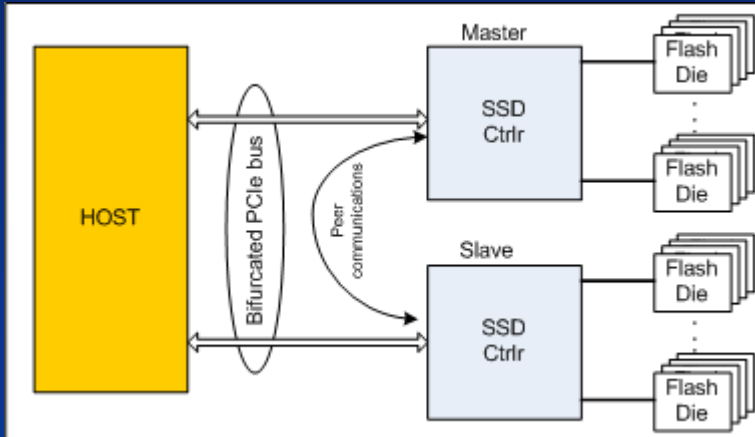
- FTL for 32TB drive may still “barely” fit in 32GB depending on the ECC scheme
 - $ECC\ overhead + (Map\ Addr\ Bits - 32)/32 + (misc\ FTL\ structures\ and\ code\ space\ memory\ fraction)$ must be $< OP$
 - SECDED implementations might work up to a point, but they are expensive and add routing complexity...
 - DDR frequencies will likely be lower – you either need more IO (routing) or more ranks...
 - No solution is very attractive, or cheap...
- TTR
 - Typically full map must be recovered prior to drive going ready. 32GB of map can take significant time to repair depending on map flush and journaling frequency and scheme
 - Harken back to the ECC discussion. Each 33 bit map entry repair could require 64B*2 DDR accesses depending on the ECC scheme
- Value Prop?
 - Not great. DDR cost, routing cost, lost performance and TTR to recover

FTL Attribute	Score
Write Amp	😊 Good
IOPs & Latency	😬😬 Read/Write slower
DDR	😬 Slow and/or Expensive
> 32bit addressing	🤔 Needs HW assists
TTR	😭 Bad
PFAIL	😊 Good
FW Complexity	🤔 More complex (TTR)



Multi-Chip Solution

- Not exactly an FTL option per se, but deserves mention
 - Basically SSD based HW RAID
 - Master/Slave Architectures could be interesting and cheaper (no switch), especially with link bifurcation
- Value Prop?
 - Depends. Bifurcation can get rid of the switch cost, but you still have additional chip and memory costs...











FTL Attribute	Score
Write Amp	😊 Good
IOPs & Latency	😊😊 Good
DDR	😊 Good
> 32bit addressing	😊 Good
TTR	😊 Good
PFAIL	😐 Neutral
FW Complexity	🤔 More complex



Host Based Map

- Many permutations and combinations for FTL design
 - Most likely hierarchical map, possibly object based. Easier to implement map flushing and coherency mechanisms. Eases memory requirements as well.
- Host resource impacts
 - Some cache and tlb impact from FTL, but should be arguably small (depending on translation implementation)
 - Main impacts are DDR consumption and additional bus overhead from FTL operations to drive
 - CPU utilization also a factor...
- TTR
 - Really depends on the journaling and recovery scheme
- Value Prop?
 - Depends. Using host resources adds complexity to the calculus...

FTL Attribute	Score
Write Amp	 Good
IOPs & Latency	  Reasonable to Good
DDR	 Mixed bag
> 32bit addressing	 More CPU intensive
TTR	 Unknown
PFAIL	 Unknown
FW Complexity	 Depends



Multi-Level Map *

- Host LBA is used to index into a First Level of map cached in DDR which references the location of a Second Level Map page which is either cached in DDR or stored in Flash
 - Flexibility to fit into a variety of DDR footprints
 - Amount of DDR employed determines what percentage of Second Level Map can be cached
 - Have to keep second level map page size small to keep system WA under control
 - If Second Level Map Page have 256 entries at 33 bits each...
 - WC FTL WA is $(256 * 33 / 8) / 4096 = \sim .26$
 - 16TB requires roughly $4.0E+9$ map entries
 - $4.0E+9 / 256 = \sim 16M$ First Level Map Entries
- TTR
 - Deadline based flushing of dirty Second Level Map pages and the First Level Map trade off recovery time vs. FTL Write Amplification
- Value Prop?
 - Good – scales to selectable DDR footprint which can work in various Form Factors and price points

FTL Attribute	Score
Write Amp	😊 Good
IOPs & Latency	😬😊 Read/Write slightly slower
DDR	😊 Good
> 32bit addressing	🤔 Needs HW assists
TTR	😊 Good
PFAIL	😊 Good
FW Complexity	🤔 More complex (TTR)

* US Patent 9,213,633



So what would a multi-level Map design look like?

- Map footprint would scale to multiple DDR footprints
 - First Level Map always cached in DDR
 - Some portion of second level map cached in DDR
- Deadline based journaling scheme
 - First level map journals in chunks
 - Velocity selected to achieve WA goals and recovery time requirements
 - Second level map journaled out by age or pfail requirements
- Hardware assists
 - Definitely for Map lookups



Conclusion

- Of all the solutions, a good multi-level map design provides the most flexibility for scaling, cost and performance
 - Possible to make Host or SSD based
 - Both have complexities
 - Both need HW assists for efficient operation



Flash Memory Summit

Thank You!